IN THE CLAIMS

The pending claims remain as follows:

1. (Original) A method, comprising:

modifying source code, including multiple instructions, by performing a Lexical Functional Grammar Analysis (LFGA) operation on one or more instructions as a function of a Digital Signal Processor (DSP) architecture; and

generating assembly code using the modified source code.

2. (Original) The method of claim 1, wherein modifying the source code by performing Lexical Functional Grammar Analysis operation further comprises:

analyzing the modified source code for syntax and semantic; and

further modifying the source code based on the outcome of the analysis.

3. (Original) The method of claim 2, further comprises:

changing structure of the modified source code through a series of iterations using Finite State Morphology (FSM) to generate efficient source code.

4. (Original) The method of claim 3, wherein modifying the structure of the modified source code using Finite State Morphology comprises:

generating intermediate code by rearranging concurrent distributed instructions based on Digital Signal Processor resources using Petri Nets algorithm; and

generating the efficient code by selecting and comparing one or more instructions in the intermediate code to one or more other similar available instructions using Genetic algorithm.

5. (Original) The method of claim 4, wherein the Digital Signal Processor resources are selected from the group consisting of registers, pipeline structures, instruction scheduling, memory, and MAC units.

6. (Original) The method of claim 4, wherein selecting and comparing one or more instructions in the intermediate code using the Genetic algorithm comprises:

selecting an instruction in the intermediate code to generate the efficient code;

computing a current number of clock cycles required to execute the selected instruction;

computing reduction in number of clock cycles by using the current and previous computed number of clock cycles;

comparing the reduction in the number of clock cycles with a predetermined reduction in number of clock cycles;

if the reduction in the number of clock cycles is greater than the predetermined reduction in the number of clock cycles, then selecting a relevant instruction based on specific Digital Signal Processor architecture and applying cross over for the selected instruction and repeating the computing number of clock cycles, the computing reduction in number of clock cycles, and the comparing operations until the reduction in the number of clock cycles is less than or equal to the predetermined reduction in the number of clock cycles; and

if the reduction in the number of clock cycles is less than the predetermined reduction in the number lock cycles, then selecting the instruction for the efficient code and repeating the above operations for a next instruction in the intermediate code.

7. (Original) The method of claim 4, wherein modifying the structure of the modified source code using Finite State Morphology further comprises:

selecting one or more instructions from the multiple instructions that have similar available instruction sets; and

performing dynamic instruction replacement on the one or more selected instructions.

8. (Original) A method of generating assembly code, comprising:

receiving a program in higher level language including multiple instructions;

parsing the program by performing Lexical Functional Grammar Analysis operation on one or more instructions such that the program complies with a specific Digital Signal Processor architecture; and

generating assembly code, for execution on a DSP, using the parsed program.

9. (Original) The method of claim 8, wherein the Digital Signal Processor architecture comprises execution resource constraints selected from the group consisting of registers, pipeline structures, instruction scheduling, memory, ALUs (Arithmetic Logic Units), and MAC (Multiply Accumulate) units.

10. (Original) The method of claim 8, wherein parsing one or more instructions in the program further comprises:

analyzing the parsed program for syntax and semantic; and

updating the parsed program for syntax and semantic based on the outcome of the analysis.

11. (Original) The method of claim 10, further comprises:

generating intermediate code by rearranging concurrent distributed instructions based on specific Digital Signal Processor architecture using Petri Nets algorithm;

generating first efficient code by selecting and comparing one or more instructions in the intermediate code to one or more other similar available instructions using Genetic algorithm; and

generating the assembly code using the first efficient code.

12. (Original) The method of claim 11, further comprises:

selecting one or more instructions from the multiple instructions that have similar available instruction sets in the first efficient code;

generating second efficient code by performing dynamic instruction replacement on the one or more selected instructions; and

generating the assembly code by mapping the second efficient code to assembly language code.

13. (Original) A method of generating assembly code for execution by a DSP, comprising:

receiving source code in higher level language including multiple instructions;

parsing the source code using Lexical Functional Grammar Analysis to modify one or more instructions in the program such that the modified instructions comply with specific Digital Signal Processor resources;

analyzing the parsed source code for syntax;

updating the parsed source code for syntax based on the analysis;

generating intermediate code by rearranging concurrent distributed instructions based on the specific Digital Signal Processor resources using Petri Nets algorithm;

generating first efficient code by selecting and comparing one or more instructions in the intermediate code to one or more other similar instructions, available to process on the specific DSP, using Genetic algorithm;

selecting one or more instructions from the multiple instructions that have similar available instruction sets in the first efficient code;

generating second efficient code by performing dynamic instruction replacement on the one or more selected instructions; and

generating the assembly code by mapping the second efficient code to assembly language code.

14. (Original) The method of claim 13, wherein the specific Digital Signal Processor resources are selected from the group consisting of registers, pipeline structures, instruction scheduling, memory, ALUs, and MAC units.

15. (Original) The method of claim 13, wherein selecting and comparing one or more instructions in the intermediate code using the Genetic algorithm comprises:

selecting an instruction in the intermediate code to generate the efficient code;

computing a current number of clock cycles required to execute the selected instruction;

computing reduction in number of clock cycles by using the current and previous computed number of clock cycles;

comparing the reduction in number of clock cycles with a predetermined reduction in number of clock cycles;

if the reduction in the number of clock cycles is greater than the predetermined reduction in the number of clock cycles, then selecting another instruction similar to the selected instruction based on the specific Digital Signal Processor architecture and applying cross over for the selected instruction and repeating the computing number of clock cycles, the computing reduction in number of clock cycles, and the comparing operations until the reduction in the number of clock cycles is less than or equal to the predetermined reduction in the number of clock cycles; and

if the reduction in the number of clock cycles is less than the predetermined reduction in the number of clock cycles, then selecting the instruction for the efficient code and repeating the above operations for a next instruction in the intermediate code.

16. (Original) A compiler for developing assembly code, comprising:

an input module to receive source code including multiple instructions;

an Lexical Functional Grammar Analysis module to modify the source code by performing an Lexical Functional Grammar Analysis operation on one or more instructions as a function of a specific Digital Signal Processor architecture; and

an output module to generate assembly code using the modified source code.

17. (Original) The compiler of claim 16, wherein the Lexical Functional Grammar Analysis module to parse the source code by performing lexical analysis on one or more instructions as a function of the specific Digital Signal Processor architecture.

18. (Original) The compiler of claim 17, wherein the Lexical Functional Grammar Analysis module further comprises:

a syntax and semantic analyzer to analyze the parsed source code for syntax and semantic and to further modify the source code based on the analysis.

19. (Original) The compiler of claim 18, wherein the compiler further comprises:

an Finite State Morphology module to generate intermediate code by allocating Digital Signal Processor resources to one or more instructions using Petri Nets algorithm, wherein the Finite State Morphology module to generate first efficient code by selecting and comparing one or more instructions in the intermediate code to one or more other similar available instructions using Genetic algorithm, and wherein the Finite State Morphology module to select one or more instructions from the multiple instructions that have similar available instruction sets in the first efficient code and to further generate second efficient code by performing dynamic instruction replacement on the one or more selected instructions, wherein the output module to generate assembly code by mapping the second efficient code to the assembly code.

20. (Original) The compiler of claim 19, wherein the Digital Signal Processor resources are selected from the group consisting of registers, pipeline structures, instruction scheduling, memory, and MAC units.

21. (Original) The compiler of claim 20, further comprises a database to store the Digital Signal Processor resources.

22. (Original) An article comprising a computer-readable medium which stores computer-executable instructions, the instructions causing a computer to:

receiving a program in higher level language including multiple instructions;

parsing the program by performing Lexical Functional Grammar Analysis operation on one or more instructions such that the program complies with a specific Digital Signal Processor architecture; and

generating assembly code, for execution on a DSP, using the parsed program.

23. (Original) The article comprising the computer-readable medium which stores computer executable instruction of claim 22, wherein parsing the program using the Lexical Functional Grammar Analysis further comprises:

analyzing the generated assembly code for syntax and semantic; and

modifying the generated assembly code based on the analysis.

24. (Original) The article comprising the computer-readable medium which stores computer executable instruction of claim 22, wherein parsing the program using the Lexical Functional Grammar Analysis further comprises:

modifying structure of the parsed program through a series of iterations using Finite State Morphology (FSM) to generate efficient source code.

25. (Original) The article comprising the computer-readable medium which stores computer executable instruction of claim 24, wherein modifying the structure of the program using the Finite State Morphology further comprises:

generating intermediate code by rearranging concurrent distributed instructions based on specific Digital Signal Processor architecture using Petri Nets algorithm;
generating first efficient code by selecting and comparing one or more instructions in the intermediate code to one or more other similar available instructions using Genetic algorithm; and

generating the assembly code using the first efficient code.

26. (Original) The article comprising the computer-readable medium which stores computer executable instruction of claim 25, wherein generating the assembly code further comprises:

selecting one or more instructions from the multiple instructions that have similar available instruction sets in the first efficient code;

generating second efficient code by performing dynamic instruction replacement on the one or more selected instructions; and

generating the assembly code by mapping the second efficient code to the assembly language code.

27. (Original) A system comprising:

a bus;

a Digital Signal Processor coupled to the bus;

a memory coupled to the DSP;

a network interface coupled to the Digital Signal Processor and the memory, to receive source code including multiple instructions; and

a compiler to modify the source code by performing an Lexical Functional Grammar Analysis operation on one or more instructions as a function of a specific Digital Signal Processor architecture and to generate assembly code using the modified source code.

28. (Original) The system of claim 27, wherein the compiler further comprises:

an Lexical Functional Grammar Analysis module to parse the source code by performing Lexical Functional Grammar Analysis operation on one or more instructions as a function of the specific Digital Signal Processor architecture.

29. (Original) The system of claim 28, wherein the Lexical Functional Grammar Analysis module further comprises:

a syntax and semantic analyzer to analyze the parsed source code for syntax and semantic and to modify the parsed source code based on the analysis.

30. (Original) The system of claim 29, wherein the compiler further comprises:

an Finite State Morphology (FSM) module to generate intermediate code by allocating Digital Signal Processor resources to each instruction in the parsed source code using Petri Nets algorithm, wherein the Finite State Morphology module to generate first efficient code by selecting and comparing each instruction in the intermediate code to one or more other similar available instruction using Genetic algorithm, and wherein the Finite State Morphology module to select one or more instructions from the multiple instructions that have similar available instruction sets in the first efficient code and to generate second efficient code by performing dynamic instruction replacement on the one or more selected instructions, wherein the compiler to generate assembly code by mapping the second efficient code to the assembly code.